Apple Assembly Line

$2.40

Volume 8 -- Issue 5 February, 1988

Peeking Inside AppleWorks 1.3, Part 3:
Keyboard Input Subroutines. 2
Percentage Printer 20
Another Quick Two-Digit Decimal Printer. 24
New Versions of S-C Word Processor 25
Printing the ProDOS Date and Time. 27

Another Way to Assemble into SYS Files

In the August 1987 issue of AAL I told how to patch the ProDOS version of S-C Macro Assembler so that target files (.TF directive) could be type SYS rather than BIN. Unfortunately the patches only work if the file is not already on the disk. In order to avoid an error message you can simply delete the target file before each assembly, and the easiest way to do this is by making an EXEC file. For example, I was working on a large program. SC.ACF is the file with a lot of include (.IN) directives. SCWP.SYSTEM is the target file. With the following three lines of text in an EXEC file named ASM, I can do an assembly by merely typing "-ASM":

```
DELETE SCWP.SYSTEM
LOAD SC.ACF
ASM
```

Unless the assembler is patched, SCWP.SYSTEM will be a type BIN file after assembly. With the following three lines in an EXEC file named SYS, I can change the filetype of SCWP.SYSTEM from BIN to SYS by typing "-SYS":

```
VERIFY SCWP.SYSTEM
$800:A9 07 8D B4 BE A9 FF 8D B8 BE 20 00 BF C3 B4 BE 4C DA FD
MGO$800
```

The VERIFY command reads the file info into a buffer at $BEB4. The code being poked into $800 changes the buffer and does an MLI call to SET FILE INFO. It then prints out the error code returned, which we hope is 00. If not 00, there was an error.

Peeking Inside AppleWorks 1.3, Part 3:
Keyboard Input Subroutines....Bob Sander-Cederlof

In this, the third of the series delving into the AppleWorks code, we will look into the low-level keyboard input code and a few associated routines. It is probably not extremely exciting, but it is fundamental, and just the right amount of code for this issue of AAL.

This time I decided to give an exact disassembly, rather than rewriting routines to my own liking as I worked through them. Nevertheless, I could not resist showing you my revisions of two subroutines, as you will see below. Until now, I have been working from a "raw" disassembly listing: that is, one produced with the Apple monitor "L" command, and my pencil. After figuring out what I wanted to reproduce, I sat down to the keyboard and typed in an equivalent program. This time I used the S-C DisAssembler to produce a set of source files, and then worked them over with the editing facilities of the S-C Macro Assembler. The result will re-assemble to an exact copy of APLWORKS.SYSTEM (version 1.3). Finally, I lifted out the subroutines I wanted to discuss this month and put them together in one source file. The result is on the following pages.

The S-C DisAssembler made my work a lot easier, but it did not do everything. It does work from a script which can detail which address ranges are code, and which are data. The version I was using also could differentiate between hex strings, ASCII strings, and address tables. Furthermore, SCDA lets me give label names in advance when I know what I want to call them. However, SCDA does not handle all the parameters following JSR XXXX lines which are part of the AppleWorks code. It does handle JSR $BF00, which is the ProDOS MLI call, but not the multitude of AppleWorks calls which I discussed in the December 1987 issue of AAL. SCDA also does not automatically generate local labels within subroutines. All labels not already given names are generated as "I.xxxx", where xxxx is the hexadecimal address. I have gone through the source code and replaced these with either meaningful names or local labels. I did leave one "I." label, because I did not know what to call it, in line 1490. SCDA also generates "X." labels for variables or subroutines outside the range being disassembled, and "Z." labels for page-zero references. I have left some of these in that form, pending more information about what they ought to be called.

If you look at the listing that follows, you may notice some strange comment lines that are filled with hexadecimal numbers. For example, see line 1640:

```
1640 * (1D35) 1066 1192 137F 193F 19C5 1BFD
```

These lines were produced by the S-C DisAssembler, and are embedded cross-reference lines. The number in parentheses is the address of the label which follows on the next line, in this case "AW.KEYIN". The list of numbers after the parentheses are addresses of instructions which refer to

S-C Macro Assembler Version 2.0 . . . . . DOS $100, ProDOS $100, both for $120
Version 2.0 DOS Upgrade Kit for 1.0/1.1/1.2 owners $20
ProDOS Upgrade Kit for Version 2.0 DOS owners $30
Cross Assemblers for owners of S-C Macro Assembler . . . . . $32.50 to $50 each
(Available: 6800/1/2, 6301, 6805, 6809, 68000, Z-80, Z-8, 8048, 8051, 8085, 1802/4/5, PDP-11, GI1650/70, Mitsubishi 50740, others)
Source Code of any S-C Macro Assembler or Cross Assembler . . . . . each, additional $100
S-C DisAssembler (ProDOS only) . . . . . without source code $30, with source $50
RAK-Ware DISASM (DOS only) . . . . . without source code $30, with source $50
ProVIEW (ProDOS-based disk utility program) $20
Full Screen Editor for S-C Macro (with complete source code) $49
S-C Cross Reference Utility . . . . . without source code $20, with source $50
S-C Word Processor, both DOS & ProDOS, both 40- & 80-columns, with complete source code $50
DP18 and DPFP, double precision math for Applesoft, including complete source code. . . $50
ES-CAPE (Extended S-C Applesoft Program Editor), including Version 2.0 With Source Code . $50
ES-CAPE Version 2.0 and Source Code Update (for Registered Owners) $30
Bag of Tricks 2 (Quality Software) . . . . . ($49.95) $45 *
S-C Documentor (complete commented source code of Applesoft ROMs) $50
Copy II Plus (Central Point Software) . . . . . ($39.95) $30 *
AAL Quarterly Disks . . . . . each $15, or any four for $45

(All source code is formatted for S-C Macro Assembler. Other assemblers require some effort to convert file type and edit directives.)

Vinyl disk pages, 6"x8.5", hold two disks each . . . . . 10 for $6 *
Diskette Mailing Protectors (hold 1 or 2 disks) . . . . . 40 cents each
(Cardboard folders designed to fit 6"X9" Envelopes.) or $25 per 100 *
Envelopes for Diskette Mailers . . . . . 6 cents each

Sider 20 Meg Hard Disk, includes controller & software . . ($695) $550 +
Sider 40 Meg Hard Disk, includes controller & software . . ($995) $860 +

Minuteman 250 Uninterruptible Power Supply . ($359) $320 +
Minuteman 300 Uninterruptible Power Supply . ($549) $490 +

65802 Microprocessor, 4 MHz (Western Design Center) . . . . . $25 *
quikLoader EPROM System (SCRG) . . . . . ($179) $170 *
PROmGRAMER (SCRG) . . . . . ($149.50) $140 *

"Exploring the Apple IIgs"	Gary B. Little	($22.95)	$21 *
"Apple IIgs Technical Reference"	Michael Fischer	($19.95)	$19 *
"65816/65802 Assembly Language Programming".	Michael Fischer	($21.95)	$20 *
"Programming the 65816"	David Eyes & Ron Lichty	($22.95)	$21 *
"Programming the Apple IIgs in C and Assembly Language". . . .	Mark Andrews	($18.95)	$18 *
"Apple //e Reference Manual".	Apple Computer	($24.95)	$23 *
"Apple //c Reference Manual".	Apple Computer	($24.95)	$23 *
"ProDOS-8 Technical Reference Manual".	Apple Computer	($29.95)	$27 *
"ProDOS-16 Technical Reference Manual"	Apple Computer	($29.95)	$27 *
"Apple IIgs Firmware Reference".	Apple Computer	($24.95)	$23 *
"Apple IIgs Hardware Reference".	Apple Computer	($24.95)	$23 *
"Apple IIgs Toolbox Reference, Volume 1".	Apple Computer	($26.95)	$24 *
"Apple IIgs Toolbox Reference, Volume 2".	Apple Computer	($26.95)	$24 *
"Apple IIgs Programmer's Introduction"	Apple Computer	($32.95)	$30 *
"ProDOS Inside and Out"	Dennis Doms & Tom Weishaar	($16.95)	$16 *
"Beneath AppleProDOS".	Don Worth & Pieter Lechner	($19.95)	$18 *
"Beneath Apple DOS".	Don Worth & Pieter Lechner	($19.95)	$18 *
"Inside the Apple //c".	Gary B. Little	($19.95)	$18 *
"Inside the Apple //e".	Gary B. Little	($19.95)	$18 *
"Understanding the Apple //e"	Jim Sather	($24.95)	$23 *
"Understanding the Apple II".	Jim Sather	($22.95)	$21 *
"Apple II+/IIe Troubleshooting & Repair Guide".	Brenner	($19.95)	$18 *
"Assembly Language for Applesoft Programmers"	Finley & Myers	($18.95)	$18 *
"Now That You Know Apple Assembly Language".	Jules Gilder	($19.95)	$18 *
"Enhancing Your Apple II, vol. 1"	Don Lancaster	($15.95)	$15 *
"Enhancing Your Apple II, vol. 2"	Don Lancaster	($17.95)	$17 *
"Assembly Cookbook for the Apple II/IIe".	Don Lancaster	($21.95)	$20 *
"Microcomputer Graphics"	Roy E.Myers	($14.95)	$14 *
"Assembly Lines — the Book".	Roger Wagner	($19.95)	$12 *

* These items add $2 for first item, $.75 for each additional item for US shipping.
\+ Inquire for shipping cost.
Customers outside USA inquire for postage needed.
Texas residents please add 8% sales tax to all orders.
<< Master Card, VISA, Discover and American Express >>

S-C Software Corporation
2331 Gus Thomasson #125
DALLAS, TX 75228
Phone 214-324-2050

AW.KEYIN. In this program, they just happen to all be JSR or JMP instructions. The disassembler produces lines like this for every label, and I left in the important ones.

Another thing you will notice in the listing is the heavy use of .PH and .EP directives. Each subroutine is surrounded by a pair of these. I wanted the listing here to show the same addresses as I found inside AppleWorks, and the .PH directive lets me do so. The object code that is stored in RAM during assembly of this source code will not be useful, because it will not be located in the addresses shown; rather, it will be all packed together starting at $0800, the default object code address. But I do not intend to use the code in this form, just display it. If you want to use one or more of these subroutines, include them in your own code WITHOUT the .PH and .EP directives.

In order to assemble these subroutines without the rest of the body of APLWORKS.SYSTEM, I had to define some subroutines not included here. Lines 1450-1470 show these names. DISPLAY.STRING and BASE.CALC.A were included in the January 1988 issue of AAL. I intend to include PRINTER.DRIVER in a future issue.

The main subroutine I want to talk about this issue begins at line 1510, and I have called it AW.KEYIN. As line 1640 shows, this is called from a lot of places; its main purpose is to get the next character from the keyboard. It also does a lot of other stuff, as we will see.

I broke the KEYIN subroutine into four parts. The first part is lines 1640-1720, and is the only entry point used by AppleWorks. The second part (lines 1730-2570) handles the actual character input, getting a character either from the type-ahead buffer or from the keyboard. The third part (lines 2580-3560) checks for certain special characters and acts on them. The last part (lines 3570-3630) stores the character in $84 and returns.

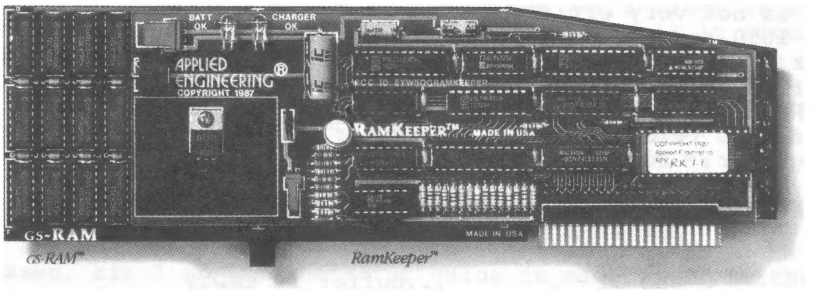
Looking at the first section first, notice lines 1660-1680. For reasons I have not yet determined, a flag is kept in pagezero location $A4 which can cause all keyboard input to be bypassed. You may remember, this same location also controls screen display (see the Jan 88 article). In both cases, if $A4 is non-zero, the operation is bypassed. You will get no display on the screen, and no keystrokes will be waited for. KEYIN will tell the world you typed an ESCAPE key, no matter what you may have really done. I don't know why all this is here yet.

Assuming $A4 is zero, Lines 1690-1720 will save the current cursor position. Apparently the cursor position may be changed in some circumstances within the KEYIN routine, and so we save them for later restoration.

The second part of KEYIN gets the next character. The "next character" may have already been typed a while ago, and tucked away in the type-ahead buffer. If so, KEYIN will get it from there. If not, it will wait until you type one.

RamKeeper™

For the "Instant On" Apple IIGS.



Permanent Storage with an "Electronic Hard Disk"

Now when you turn on your IIGS your favorite program can appear on screen in just a few seconds! With RamKeeper, your IIGS memory card will retain stored programs and stored data while your IIGS is turned off. RamKeeper allows you to divide your IIGS memory into part "electronic hard disk" and part RAM for your programs workspace—in almost any way you want and at anytime you want. GS-RAM, GS-RAM Plus, Apple IIGS memory card and most other IIGS memory cards are compatible with RamKeeper.

Supports Up to Two IIGS Memory Cards at the Same Time

If you bought your IIGS with Apple's memory card and later wished you had the GS-RAM, no problem. RamKeeper will support both cards plugged into RamKeeper simultaneously!

How it Works

Just unplug your IIGS memory card from your computer, plug your IIGS memory card into RamKeeper, plug RamKeeper into the IIGS memory slot. If you have another IIGS memory card, an additional card socket on RamKeeper will accommodate your second card. That's all there is to it!

"I've purchased several Applied Engineering products over the years. They're always well made and performed as advertised. I recommend them wholeheartedly."

Steve Wozniak, the creator of Apple Computer

Reliability from the Most Experienced

Applied Engineering has the most experience in the industry with battery-backed memory for the Apple so you are assured of the most reliable memory back-up system available. And in the world of battery-backed memory, reliability is everything! That's why Applied Engineering uses the more dependable Gel-Cell's instead of Ni-Cad batteries (if Ni-Cad's aren't discharged periodically, they lose much of their capacity). RamKeeper has close to 6 times (about 6 hours) the "total power failure" back-up time of other systems. When power returns, RamKeeper automatically re-charges the battery to a full charge. With power from your wall outlet, RamKeeper will back-up your IIGS memory cards RAM indefinately.

RamKeeper Has and Does It All!

- Allows instant access to your programs without slow disk delays
- Configure Kilobytes or Megabytes of instant ROM storage for your favorite programs
- Reduces power strain to your internal IIGS' power supply
- Contains back-up status L.E.D.'s
- Can support up to two IIGS memory cards simultaneously
- Supports both 256K installed memory chip boards like GS-RAM and the Apple IIGS Memory Expansion Card as well as 1 MEG installed memory chip boards like GS-RAM Plus
- 5-year hassle-free warranty
- 15 day money back guarantee
- Proudly made in the U.S.A.
- RamKeeper comes *complete* with battery, software and documentation

Only $179.00!

(GS-RAM card shown in photo not included)

Order Your RamKeeper Today!

See your dealer or call (214) 241-6060, 9:00-11:00 CST, 7 days a week, or send check or money order to Applied Engineering. MasterCard, VISA and C.O.D. welcome. Texas residents add 7% sales tax. Add $10.00 if outside U.S.A.

APPLIED ENGINEERING™

The Apple enhancement experts.

(214) 241-6060

P.O. Box 798, Carrollton, TX 75006

Prices subject to change without notice.

You will recall that I talked about the POLL.KEYBOARD subroutine and the type-ahead buffer in the December 1987 issue. Lines 3970-4300 of this month's code show two more subroutines associated with the type-ahead buffer. CLEAR.KEYBUF will empty the type-ahead buffer, and CHECK.KEYBUF will test whether there are any characters in the buffer or not. For some reason there does not appear to be any subroutine to get a character out of the type-ahead buffer; instead, that code is included in-line wherever it is needed. This is not very efficient, but it "is what is". Lines 1760-1850 are one example of this. If it were up to me, I think I would have written one subroutine which returned Carry Clear if there were no characters in the buffer, or Carry Set if there was a least one character. In the latter case, I would also return the next character from the buffer in the A-register. Something like this:

```
GET.CHAR.FROM.KEYBUF.IF.ANY
        LDA KEYBUF.OUT
        CMP KEYBUF.IN
        BEQ .2              ...buffer is empty
        TAX                 Use the "out" index
        LDA KEYBUF,X        Get character from buffer
        INX                 Advance buffer index
        CPX #10             Compare to buffer size
        BCC .1              ...not off the end yet
        LDX #0              Wrap-around to beginning
.1      STX KEYBUF.OUT      New "out" index
        SEC                 Indicate we got a character
        RTS
.2      CLC                 Indicate buffer was empty
        RTS
```

If that subroutine existed, I could replace lines 1760-1850 with

```
  1760 JSR GET.CHAR.FROM.KEYBUF.IF.ANY
  1770 BCS .11     ...got a char
```

and eliminate the CHECK.KEYBUF subroutine. Cleaner code, in my estimation, and shorter too.

If there is no character waiting in the type-ahead buffer, lines 1860 and following will get one from the keyboard. This is not as simple as it sounds, because there are a lot of options. First, there are three possible choices for the kind of cursor display during keyboard input. As the comments in lines 1860-1940 say, you may choose a blinking underline cursor, a flashing block cursor, or no cursor at all. If you are an AppleWorks user, you will recognize the first two options as the insert and overwrite cursors. The third, no cursor at all, used when a message like "Press Any Key to Continue" is displayed. The options are selected by values in two variables I have named KEYIN.CURSOR.FLAG and KEYIN.CURSOR.TYPE.

Just in case we are going to display a cursor, lines 1960-2050 retreive the current screen character at the cursor position and save it. If the current cursor position is in AUX RAM, then screen memory is left switched in the AUX position. Line 2540 will later switch it back to MAIN RAM, after a keystroke has been read.

Lines 2070-2080 test whether we have chosen to display a cursor or not. If not, lines 2090-2140 will call on the READ.KEYBOARD subroutine to wait for you to type something. READ.KEYBOARD (lines 3670-3950) is a little smarter than the average keyboard reader. First, it does not wait forever. You call it with a timeout value in the Y-register. The loop which polls the keyboard counts down the timeout value. If it reaches zero, READ.KEYBOARD returns with a FALSE status; if you type a key before timeout, READ.KEYBOARD returns with the character in the A-register and a TRUE status. In addition, notice at line 3750 that READ.KEYBOARD stores whatever character was in the A-register on the screen. This character is determined by your choice of a cursor display. Finally, this subroutine reads the Open- and Solid-Apple keys. If either one of them is depressed, bit 7 of the character value is made 1; if neither is depressed, bit 7 is made 0.

If you do want a cursor display, lines 2150-2170 decide which type you want. Lines 2180-2310 handle the blinking underline, and lines 2320-2460 handle the flashing block. The blinking underline is in reality a repeating series of three characters: an underline, a blank, and the original screen character. The timeout values are chosen so that the underline and blank each are on the screen for about 17% of the time, and the original character the remaining 66%. If you like changing such things, this is the place to do it. You can change the overall speed of the blink, or change the ratio, or change the characters. You might choose one of the mousetext characters here, just for fun, instead of the underline. For example, $5C instead of $DF would display an under- and over-line character.

The flashing block cursor display involves change the character retrieved from the screen to the inverse equivalent character, and alternating between it and the original character. Lines 2330-2420 put up the inverse character for 79% of the time, and lines 2430-2460 put up the normal character for 21% of the time.

Regardless of which of the two displays you choose, when you type a key control will go to ".10" at line 2480. Here the original screen character will be restored.

And, regardless of which of three cursor-types, you eventually wind up at ".11", line 2540. Here we turn screen RAM back to MAIN, and test whether you have opted for further analysis of the input character or not. The variable at $FC4, which I did not give a meaningful name yet, controls that option. If $FC4 is non-zero, the game is all over; if zero, the KEYIN.ANALYSIS section gets to do some work.

I have detailed what KEYIN.ANALYSIS does in the comments in lines 2580-2790. Any normal printing character will simply be passed to you without further action. Any control character will clear the type-ahead buffer and then be passed to you. Any Apple-character will also clear the type-ahead buffer, and if it is not one of the special ones it will be passed to you.

The "special" Apple-characters are what KEYIN.ANALYSIS is here for. Lines 2990-3040 map lower-case letters to the upper-case range. Apple-E is used to change the cursor-type. If you have a blinking underline cursor on the screen, typing Apple-E makes it change to a flashing block cursor. And vice versa. The cursor-toggling code is in lines 3500-3560. Apple-H is used to print the screen contents on your printer. The code to do the screen dump is located right here, in lines 3170-3400. Apple-Q, -S, and -Y are also looked for. If you type and Apple-/, lines 2900-2980 change it to an Apple-? and pass the new code on to you.

By the way, that code for the Apple-/ is sure strange. It tests for the "/" characte THREE TIMES! Why? I can think of two reasonable answers. First, maybe there used to be two other keys besides "/" which were changed to "?"; if so, maybe the author simply patched the code the way it is now rather than revising and re-assembling it. Naw.... More likely is that this is a "hook" for catching knockoffs of the code. I have found other hooks, such as deliberate use of a 3-byte reference to a pagezero variable when a 2-byte instruction could have been used. Anyone copying the code to create an illegal copy of AppleWorks for sale under their own name might miss these hooks, and find themselves waking up in court.

The Screen Print code is rather interesting. It calls on a subroutine I call PRINTER.DRIVER for printing each line, but that subroutine is not listed here. There are four different types of calls to PRINTER.DRIVER, controlled by the value in the A-register. If (A) is zero, the printer is initialized. On an Apple //e this means sending out a control-I code, and "80N", which turns off the screen echo many printer interfaces would otherwise perform. On an Apple //c it sends out the controls to set up the baud rate and LF after CR for a serial ImageWriter printer. More on this in a future issue.

Lines 3230-3290 copy a line from the screen to a buffer starting at $900, and then print it on the printer. At lines 3250-3270 PRINTER.DRIVER is called with (A)=79, the line length, which means to print a string whose address follows the JSR instruction. Then at lines 3280-3290 PRINTER.DRIVER is called with (A)=$FE, which means to print a carriage return. Finally, Lines 3340-3350 call PRINTER.DRIVER with (A)=$FF, meaning to "close" the driver.

After the screen print is completed, lines 3370-3390 make sure the cursor is back where it started. However, I did not notice anywhere it ever got moved, so this may be redundant code. The subroutine called here, MOVE.CURSOR.TO.XY, is shown in lines 4320-4480. It uses a round-about method, building a string for last month's DISPLAY.STRING subroutine. The setup for

DISPLAY.STRING is also handled in a round-about way, using the code I called POINT.PSTR.AT.0A00, lines 4500-4600. I haven't yet found any reason why this code is not simpler. Why go through the back bedroom to get from the kitchen to the dining room? I think I would have written MOVE.CURSOR.TO.XY like this:

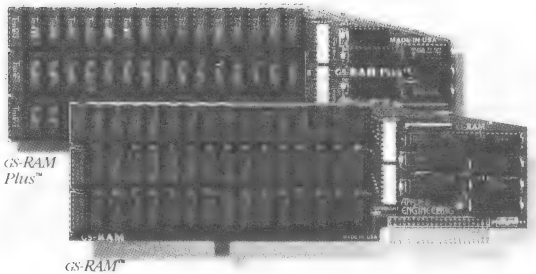
```
SC.MOVE.CURSOR.TO.XY
        TXA
        CLC
        ADC AW.LEFT
        STA AW.CH
        TYA
        ADC AW.TOP
        STA AW.CV
        RTS
```

The screen print code also calls on the subroutine I show in lines 4620-4940, COPY.SCRN.LINE.TO.0900. This subroutine picks up one line of characters, translating them from display codes to printer codes, and places the result in a buffer at $0900. Another subroutine, MAP.SCRN.CHARS.TO.INTERNAL, does the translation. That MAP... subroutine is never called from anywhere else, so it really should be considered part of the COPY... subroutine. Together they take 72 bytes. I rewrote COPY..., and my simpler, shorter version is shown in lines 5240-5600. Mine only takes 50 bytes. If someone had time to go over the whole program the same way, there might be room for a lot of new features.

```
                1000 *SAVE AW.SUBS.3
                1010 *--------------------------------
 14-            1020 AW.CH    .EQ $14
 15-            1030 AW.CV    .EQ $15
 16-            1040 AW.BASE  .EQ $16,17
 80-            1050 PSTR     .EQ $80,81
 84-            1060 Z.84     .EQ $84      Current Keyin Character
 98-            1070 PNTR     .EQ $98,99
 A4-            1080 Z.A4     .EQ $A4
                1090 *--------------------------------
 0900-          1100 X.0900 .EQ $0900     Used during Screen Print (Apple-H)
 0901-          1110 X.0901 .EQ $0901
                1120 *--------------------------------
 0A00-          1130 X.0A00 .EQ $0A00     Used for building little strings
 0A01-          1140 X.0A01 .EQ $0A01
 0A02-          1150 X.0A02 .EQ $0A02
                1160 *--------------------------------
 0EA7-          1170 X.0EA7 .EQ $0EA7
                1180 *--------------------------------
 0F3B-          1190 X.0F3B .EQ $0F3B
 0F3D-          1200 X.0F3D .EQ $0F3D
 0FC4-          1210 X.0FC4 .EQ $0FC4
                1220 *--------------------------------
                1230        .PH $1099
 1099- 00 0A    1240 HANDLE.0A00 .DA X.0A00
                1250        .EP
                1260
                1270 *--------------------------------
                1280        .PH $1176
 1176- 01       1290 KEYIN.CURSOR.TYPE   .HS 01   00=underline, 01=flashing
 1177- 01       1300 KEYIN.CURSOR.FLAG   .HS 01   00=no cursor, 01=cursor
 1178-          1310                     .BS 2    other variables
 117A-          1320 KEYBUF              .BS 10   type-ahead buffer
 1184- 00       1330 KEYBUF.IN           .HS 00
 1185- 00       1340 KEYBUF.OUT          .HS 00
                1350        .EP
                1360
```

For Those Who Want the Most. From Those Who Make the Best. GS-RAM.™

Now expand the IIGS' RAM and ROM with up to 8 MEG of "Instant On" memory with the all new GS-RAM!



GS-RAM has an all new design. A design that delivers higher performance including increased speed, greater expandability, and improved software.

More Sophisticated, Yet Easier to Use

GS-RAM works with all IIGS software. In fact *any* program that runs on Apple's smaller memory card runs on the GS-RAM. But with GS-RAM you can have more memory, improved performance, and almost unlimited expansion capabilities. We've designed the new GS-RAM to be easier to use too—you don't have to adjust the size of your RAM disk every time you use a DMA device. No other RAM card with more than 4 banks of memory installed can make the same claim.

More than Just Hardware

Each GS-RAM and GS-RAM Plus includes the most powerful set of IIGS software enhancements available anywhere. In fact, our nearest competitor offers only a fraction of the invaluable programs that we include with each GS-RAM card. This software includes the most powerful disk-caching program available, the GS-RAM Cache. The Cache will make most of your applications run up to 7 *times faster.* Also included is a diagnostic utility that lets you test your GS-RAM by graphically showing the location of any bad or improperly installed RAM chips. And for AppleWorks users, we give you our exclusive Expander program that dramatically enhances both the capabilities and speed of AppleWorks.

Making AppleWorks Even Better

Applied Engineering's Expander program eliminates AppleWorks internal memory limits allowing it to recognize up to 8 megabytes of desktop workspace. You can increase the limits from only 7,250 lines to 22,600 lines in the word processor and from 6,350 records to 22,600 records in the database. The Expander allows all of AppleWorks, including print functions, to automatically load into RAM. The clipboard size will increase from 255 to 2,042 lines maximum. GS-RAM will automatically segment larger files so you can save them onto multiple floppies. And GS-RAM provides a built-in print buffer that allows you to continue working in AppleWorks while your printer is still processing text. You can even load Pinpoint or MacroWorks and your favorite spelling checker into RAM for instant response.

Grow by Kilobytes or Megabytes

We offer GS-RAM in two configurations so you can increase your memory 256K at a time (GS-RAM) or a megabyte at a time (GS-RAM Plus). Both are IIGS compatible and both come with our powerful enhancement software. GS-RAM can hold up to 1.5 MEG of 256K chips and GS-RAM Plus can hold up to 6 MEG using 1 MEG chips. And since both use standard RAM chips (not high-priced SIMM's), you'll find expanding your GS-RAM or GS-RAM Plus easy, convenient, and very economical. For further expansion, you can plug a 2 MEG "piggyback" card into the GS-RAM's expansion port for up to 3.5 MEG of total capacity. Or up to a whopping 8 MEG on GS-RAM Plus. If a GS-RAM owner outgrows 3.5 MEG, he can easily upgrade to GS-RAM Plus for a nominal charge.

Permanent Storage for an "Instant On" Apple

With our RamKeeper™ back-up option, your GS-RAM or GS-RAM Plus will retain both programs and data while your IIGS is turned off! Now when you turn your IIGS back on, your favorite software is on your screen in under 4 seconds! With RamKeeper you can divide your IIGS memory into part "electronic hard disk," and part extended RAM. Even change the memory boundaries at any time—and in any way—you want. Because Applied Engineering has the most experience in the industry with battery-backed memory for the Apple, you are assured of the most reliable memory back-up system available. And in the world of battery-backed memory, *Reliability* is everything. That's why Applied Engineering uses state-of-the-art "GEL-CELL's" instead of Ni-Cad batteries (if Ni-Cads aren't discharged periodically, they lose much of their capacity). RamKeeper has about 6 hours of "total power failure" back-up time. That's 6 times the amount of other systems. But with power from your wall outlet, RamKeeper will back-up GS-RAM, GS-RAM Plus, or most other IIGS memory cards indefinitely. Should you ever have a "total power failure," RamKeeper switches to its 6-hour battery. When power returns, RamKeeper will automatically recharge the battery to full power. RamKeeper incorporates a dual-rate charger, status L.E.D.'s, and advanced power reducing circuitry. RamKeeper comes *complete* with battery, software, and documentation.

"In quality, performance, compatibility, expandability and support, Applied Engineering's GSRAM and GSRAM Plus are number one."

Steve Wozniak, the creator of Apple Computer

GS-RAM's Got it ALL!

- 5-year warranty — parts & labor
- 6 RAM banks (most cards have 4)
- Memory expansion port
- ROM expansion port
- Ultra-fast disk caching on ProDOS 8 *AND* ProDOS 16.
- Expands AppleWorks internal limits
- Includes hi-res self test
- No soldered-in RAM chips
- Expandable to 8 MEG
- No configuration blocks to set
- RamKeeper back-up option allows permanent storage of programs & data
- 15-day money-back guarantee
- Proudly made in the U.S.A.

GS-RAM with 256K	**$189**
GS-RAM with 512K	**$259**
GS-RAM with 1 MEG	**$399**
GS-RAM with 1.5 MEG	**$539**
GS-RAM with 2.5 to 3.5 MEG	**CALL**
GS-RAM Plus with 1-8 MEG	**CALL**
RamKeeper Option	**$179**

Order today!

See your dealer or call Applied Engineering today, 9 a.m. to 11 p.m. 7 days. Or send check or money order to Applied Engineering. MasterCard, VISA and C.O.D. welcome. Texas residents add 7% sales tax. Add $10.00 outside U.S.A.

APPLIED ENGINEERING™

The Apple enhancement experts.

(214) 241-6060

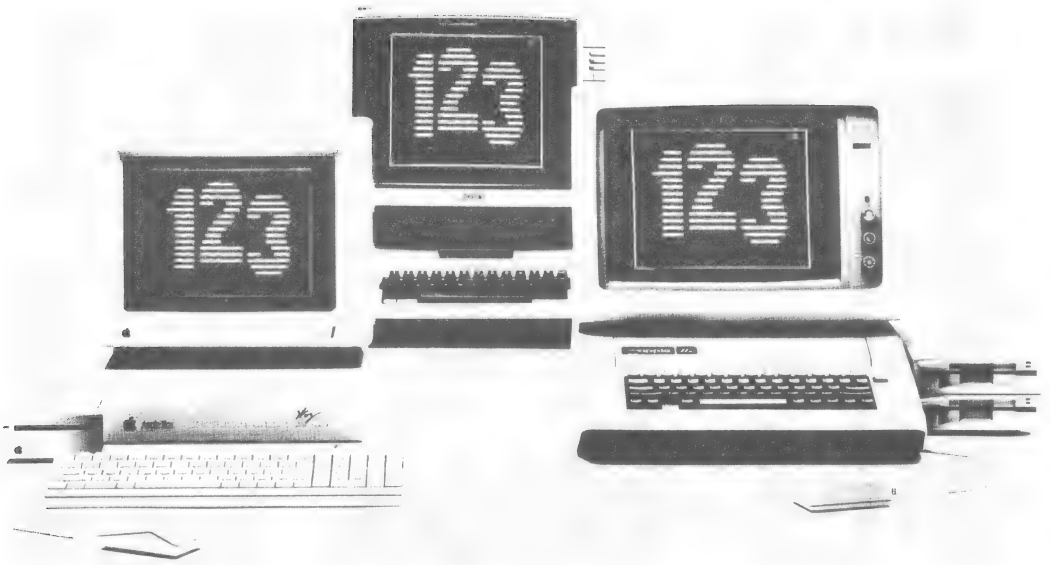
P.O. Box 798, Carrollton, TX 75006

Prices subject to change without notice.

GS-RAM, GS-RAM Plus and RamKeeper are trademarks of Applied Engineering. Other brands and product names are registered trademarks of their respective holders.

```
                    1370 *--------------------------------
C000-               1380 KEYBOARD      .EQ $C000
C010-               1390 STROBE        .EQ $C010
C054-               1400 SCRN.MAIN     .EQ $C054
C055-               1410 SCRN.AUX      .EQ $C055
C061-               1420 APPLE.OPEN    .EQ $C061
C062-               1430 APPLE.SOLID   .EQ $C062
                    1440 *--------------------------------
14D1-               1450 DISPLAY.STRING       .EQ $14D1   subroutine in AAL Jan 88
1717-               1460 BASE.CALC.A          .EQ $1717   subroutine in AAL Jan 88
1C21-               1470 PRINTER.DRIVER       .EQ $1C21   subroutine in future AAL
                    1480 *--------------------------------
1D0E-               1490 I.1D0E      .EQ $1D0E
                    1500 *--------------------------------
                    1510         .PH $1D30
                    1520 *--------------------------------
                    1530 * (1D30) 1D71 1D7B 1D8C 1D9A 1DA6 1DBC 1DCA
1D30-               1540 KEYIN.CHAR.UNDER.CURSOR   .BS 1
                    1550 * (1D31) 1D6C 1DC7 1F0F
1D31-               1560 KEYIN.COLUMN.INDEX        .BS 1
                    1570 * (1D32) 1E27 1E2A 1E3C 1E3F
1D32-               1580 KEYIN.APPLE.H.LINE.NUMBER .BS 1
                    1590 * (1D33) 1D3E 1E4E
1D33-               1600 KEYIN.CURSOR.CH           .BS 1
                    1610 * (1D34) 1D43 1E51
1D34-               1620 KEYIN.CURSOR.CV           .BS 1
                    1630 *--------------------------------
                    1640 * (1D35) 1066 1192 137F 193F 19C5 1BFD
                    1650 AW.KEYIN
1D35- A5 A4         1660         LDA Z.A4      If non-zero, exit now as if
1D37- F0 03         1670         BEQ .1        ...you typed <ESC>
1D39- 4C 7B 1E      1680         JMP KEYIN.EXIT.ESCAPE
1D3C- A5 14         1690 .1      LDA AW.CH     Save current cursor position
1D3E- 8D 33 1D      1700         STA KEYIN.CURSOR.CH
1D41- A5 15         1710         LDA AW.CV
1D43- 8D 34 1D      1720         STA KEYIN.CURSOR.CV
                    1730 *--------------------------------
                    1740 * (1D46) 1E1A 1E57 1E78
                    1750 KEYIN.ANOTHER.CHAR
1D46- 20 B2 13      1760         JSR CHECK.KEYBUF  Any characters in key-buffer?
1D49- F0 13         1770         BEQ .2            ...no
1D4B- AE 85 11      1780         LDX KEYBUF.OUT    ...yes, get char from keybuf
1D4E- BD 7A 11      1790         LDA KEYBUF,X
1D51- E8            1800         INX               Bump keybuf out-index
1D52- E0 0A         1810         CPX #10           At end of buffer?
1D54- 90 02         1820         BCC .1            ...no
1D56- A2 00         1830         LDX #$00          ...yes, wrap around to beginning
1D58- 8E 85 11      1840 .1      STX KEYBUF.OUT    Save new keybuf out-index
1D5B- 4C D0 1D      1850         JMP .11           Use the character
                    1860 *--------------------------------
                    1870 *   Key-buffer is empty, so we need to get a character
                    1880 *   directly from the keyboard.  Therefore, we must:
                    1890 *      1.  Save character now on screen under cursor
                    1900 *      2.  Put up an appropriate cursor
                    1910 *          a.  blinking underline
                    1920 *          b.  flashing screen char
                    1930 *          c.  no cursor at all
                    1940 *      3.  Get a keystroke.
                    1950 *--------------------------------
1D5E- A5 15         1960 .2      LDA AW.CV     Point to the character at the cursor
1D60- 20 17 17      1970         JSR BASE.CALC.A
1D63- A5 14         1980         LDA AW.CH
1D65- 4A            1990         LSR
1D66- B0 03         2000         BCS .3        ...Odd column, main RAM
1D68- 8D 55 C0      2010         STA SCRN.AUX ...Even column, aux RAM
1D6B- A8            2020 .3      TAY
1D6C- 8C 31 1D      2030         STY KEYIN.COLUMN.INDEX
1D6F- B1 16         2040         LDA (AW.BASE),Y
1D71- 8D 30 1D      2050         STA KEYIN.CHAR.UNDER.CURSOR
                    2060 *---Select type of cursor--------
1D74- AE 77 11      2070         LDX KEYIN.CURSOR.FLAG
1D77- D0 0C         2080         BNE .5        ...we do want a cursor display
                    2090 *---No cursor at all-------------
1D79- A0 FF         2100 .4      LDY #$FF      Long time-out count
1D7B- AD 30 1D      2110         LDA KEYIN.CHAR.UNDER.CURSOR
1D7E- 20 0C 1F      2120         JSR READ.KEYBOARD
1D81- F0 F6         2130         BEQ .4        ...No key yet
1D83- D0 4B         2140         BNE .11       ...got a keystroke!
```

```
                        2150 *---Some type of cursor----------
1D85- AE 76 11          2160 .5     LDX KEYIN.CURSOR.TYPE
1D88- D0 1C             2170        BNE .8          ...Use flashing character
                        2180 *---Use blinking underline-------
                        2190 *   Repeat loop of underline, char, blank, char
1D8A- A9 DF             2200 .6     LDA #$DF        Underline character
1D8C- CD 30 1D          2210        CMP KEYIN.CHAR.UNDER.CURSOR
1D8F- D0 02             2220        BNE .7          ...screen not now an underline
1D91- A9 A0             2230        LDA #" "        ...now underline, change to blank
1D93- A0 1C             2240 .7     LDY #28         Short time-out
1D95- 20 0C 1F          2250        JSR READ.KEYBOARD
1D98- D0 2C             2260        BNE .10         ...got a key!
1D9A- AD 30 1D          2270        LDA KEYIN.CHAR.UNDER.CURSOR
1D9D- A0 6C             2280        LDY #108        Long time-out
1D9F- 20 0C 1F          2290        JSR READ.KEYBOARD
1DA2- D0 22             2300        BNE .10         ...got a key!
1DA4- F0 E4             2310        BEQ .6          ...always (no keystroke yet)
                        2320 *---Use flashing character-------
1DA6- AD 30 1D          2330 .8     LDA KEYIN.CHAR.UNDER.CURSOR
1DA9- 29 7F             2340        AND #$7F        Change character to inverse
1DAB- C9 40             2350        CMP #$40
1DAD- 90 06             2360        BCC .9
1DAF- C9 60             2370        CMP #$60
1DB1- B0 02             2380        BCS .9
1DB3- 29 BF             2390        AND #$BF
1DB5- A0 6C             2400 .9     LDY #$6C
1DB7- 20 0C 1F          2410        JSR READ.KEYBOARD
1DBA- D0 0A             2420        BNE .10         ...got a key!
1DBC- AD 30 1D          2430        LDA KEYIN.CHAR.UNDER.CURSOR
1DBF- A0 1C             2440        LDY #28         Short time-out
1DC1- 20 0C 1F          2450        JSR READ.KEYBOARD
1DC4- F0 E0             2460        BEQ .8          ...no keystroke yet
                        2470 *---Got a key, restore scrnchar--
1DC6- 48                2480 .10    PHA
1DC7- AC 31 1D          2490        LDY KEYIN.COLUMN.INDEX
1DCA- AD 30 1D          2500        LDA KEYIN.CHAR.UNDER.CURSOR
1DCD- 91 16             2510        STA (AW.BASE),Y
1DCF- 68                2520        PLA
                        2530 *-------------------------------
1DD0- 8D 54 C0          2540 .11    STA SCRN.MAIN       Be sure in main RAM
1DD3- AE C4 0F          2550        LDX X.0FC4    Should we analyze the char?
1DD6- F0 03             2560        BEQ KEYIN.ANALYSIS    ...yes
1DD8- 4C 7D 1E          2570        JMP KEYIN.EXIT    ...no, just store and return
                        2580 *-------------------------------
                        2590 *   Analyze the Character
                        2600 *
                        2610 *   1.  If character is $20-7F, just store it and return.
                        2620 *   2.  Otherwise, start by clearing the key-buffer.
                        2630 *   3.  If character is $00-1F, it will be fall through all
                        2640 *           other tests and return after being stored.
                        2650 *   4.  If character is $80-FF, an Apple key was down.
                        2660 *   5.  If character is $E1-FA, it is lower-case and will
                        2670 *           be changed to upper-case ($C1-DA).
                        2680 *   6.  If character is not one of the following, just
                        2690 *           store it and return.
                        2700 *
                        2710 *       Apple-/   Change to Apple-?, and return.
                        2720 *       Apple-H   Print the screen, get another char.
                        2730 *       Apple-Q   Clobber Z.A4, substitute <ESC>, return.
                        2740 *       Apple-Y   Change to Control-Y and return.
                        2750 *       Apple-S   If $EA7 non-zero, just store and return.
                        2760 *                 Else, clobber Z.A4, substitute <ESC>,
                        2770 *                    and return.
                        2780 *       Apple-E   Toggle Cursor Mode, get another char.
                        2790 *-------------------------------
                        2800 KEYIN.ANALYSIS
1DDB- C9 20             2810        CMP #$20
1DDD- 90 07             2820        BCC .1          ...Control Character, analyze it.
1DDF- C9 7F             2830        CMP #$7F
1DE1- B0 03             2840        BCS .1          ...Apple Character, analyze it.
1DE3- 4C 7D 1E          2850        JMP KEYIN.EXIT
                        2860 *---Analyze the keychar----------
1DE6- 48                2870 .1     PHA             Save character temporarily...
1DE7- 20 E0 1F          2880        JSR CLEAR.KEYBUF
1DEA- 68                2890        PLA             ...and get the character back.
```



Now Apple speaks IBM. Three times faster than IBM.

Introducing PC Transporter.™ The Apple® II expansion board that lets you run MS®-DOS programs.

Now your Apple II can run over 10,000 programs you could never use before. Like Lotus® 1-2-3,® MultiMate,® dBASE III PLUS® Even Flight Simulator®

With PC Transporter, MS-DOS programs run on your Apple II like they do on IBM® PC's or compatibles. With one important difference. PC Transporter runs most of those programs *three times faster* than an IBM PC/XT.®

Plus, to speed through number-crunching tasks, you can use our optional 8087-2 math co-processor chip. It plugs into a socket on the PC Transporter.

Less expensive than an IBM clone.

Sure, a stripped-down IBM

clone costs about the same as the PC Transporter. But the peripherals it takes to get the clone up and running make the clone cost about three times what our American-made card costs.

You don't have to buy new hardware to use PC Transporter.

Works with the hardware you already own.

With PC Transporter, MS-DOS programs see your Apple hardware as IBM hardware. You can use the same hardware you have now.

With IBM software, your Apple hardware works just like IBM hardware. Including your drives, monitors, printers, printer cards, clock cards and serial clocks.

You can use your IIe® or IIGS™ keyboard with IBM software. Or use our optional IBM-style keyboard (required for the II Plus).

You can use your Apple mouse. Or an IBM compatible serial mouse.

Plenty of power.

PC Transporter gives you as much as 640K of user RAM and 128K of system RAM in the IBM mode.

PC Transporter also is an Apple expansion card, adding up to 768K of extra RAM in the Apple mode. The Apple expansion alone is a $300 value.

Easy to install.

You can install PC Transporter in about 15 minutes, even if you've never added an expansion board. You don't need special tools. Simply plug it into an Apple expansion slot (1 through 7 except 3), connect a few cables and a disk drive, and go!

PC Transporter taps into the world's largest software library. Now your Apple can run most of the IBM software you use at work. And it opens a new world of communications programs, games and bulletin boards.

A universal disk drive controller.

PC Transporter supports 3.5" and 5.25" MS-DOS and ProDOS formatted diskettes. You'll shift instantly between Apple ProDOS and IBM MS-DOS.

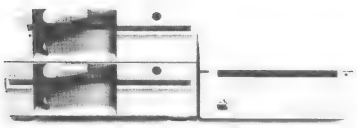
You'll need our versatile 5.25" 360K drive system to run IBM applications from 5.25" floppy disks. Use your Apple 5.25" drive for Apple 5.25" disks.

An Apple Disk 3.5 Drive will upport the new 3.5"disks whether they're IBM MS-DOS formatted or Apple ProDOS formatted. The PC Transporter acts like an Apple Disk 3.5 Drive disk controller for IIGS, IIe, and II Plus users.

PC Transporter supports up to 5 drives in a number of combinations.

For example, you can connect a 5.25 Applied Engineering 360K dual-drive system directly to the card. Then plug two daisy-chained Apple 3.5 Drives (not the Apple UniDisk 3.5) to the dual-drive system. For a fifth drive, use a ProDOS file as an IBM hard disk.

PC Transporter controls Apple and IBM compatible disk drives. It supports 3.5" and 5.25" MS-DOS and ProDOS formatted diskettes.



Versatile data storage.

PC Transporter reads MS-DOS and translates it into Apple native ProDOS. You can store IBM programs and data on any ProDOS storage device including the Apple 3.5 Drive, Apple UniDisk™ 3.5, Apple 5.25" drive, SCSI or ProDOS compatible hard drives. (You can use the Apple UniDisk 3.5 with its own controller card for storing programs and data, but not for directly booting an IBM formatted disk.)

You can even use our 360K PC compatible drive for ProDOS

Make your Apple speak IBM.

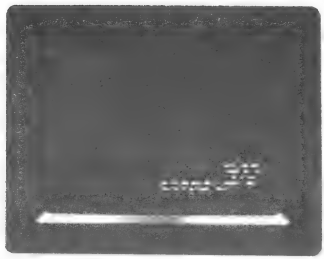
PC Transporter memory choices.

RAM in Apple mode:	RAM in IBM mode:	Price:
384K	256K	$489.00
512K	384K	529.00
640K	512K	569.00
768K	640K	609.00

Note: The IBM mode is 128K less because the PC Transporter uses 128K for system memory.

IIGS Installation Kit	**49.00**
IIe/II Plus Installation Kit	**39.00**
PC Transporter Accessories	
5.25" IBM Format 360K Drive Systems	
Single-Drive System	**269.00**
Dual-Drive System	**399.00**
Half-Height Drive (add to Single-Drive system to make Dual-Drive)	**135.00**
IBM-Style Keyboard (required for Apple II Plus. Requires IBM Keyboard Cable.)	**139.00**
IBM Keyboard Cable	**34.00**
Sony RGB Monitor	**499.00**
Analog RGB Cable (for use with Sony monitor)	**39.00**
Digital RGB Cable (for use with Sony monitor)	**39.00**
Digital RGB Adapter	**24.00**
ColorSwitch (included with IIGS Installation Kit)	**44.00**
128K ZIP	**40.00**
PC Transporter Memory Expansion Chip Set	**per set**
8087-2 Math Co-processor Chip	**229.00**
Heavy Duty Power Supply (IIe and II Plus only)	**69.00**

See your dealer or call or send check or money order to Applied Engineering. MasterCard, VISA and COD welcome. Texas residents add 6¼% sales tax.



PC Transporter produces better IBM graphics than IBM. Analog is sharper than digital. So with an analog RGB monitor, PC Transporter's CGA graphics and text are superior to IBM's digital display — even while running IBM software!

And, you can also use an Apple composite monitor in IBM text or graphics mode.

storage and a 143K Apple 5.25" drive for MS-DOS storage.

Created by Apple's original designers.

The brains behind PC Transporter were also behind your Apple II.

The PC Transporter design team includes the former project managers for the creation of the Apple IIe and IIc. The co-designer of the Apple II disk controller. And the first full-time Apple programmer and author of the ProDOS operating system.

So you know the PC Transporter and your Apple were made for each other.

Support and service from the leader in Apple add-ons.

Applied Engineering sells more Apple peripheral boards than anyone else — including Apple Computer. So you know we'll be around after the sale.

PC Transporter comes with a 15-day money back guarantee. If you're not fully satisfied after using it, return it for a full refund. PC Transporter also comes with a 1-year warranty.

How to get your PC Transporter today.

See your dealer. Or call Applied Engineering any day between 9 a.m. and 11 p.m. CST at 214-241-6060.

Applied Engineering

The Apple enhancement experts.

P.O. Box 798, Carrollton, TX 75006
214-241-6060
A Division of AE Research Corporation.

```
                    2900 *---Check for Apple-Slash--------
1DEB- C9 AF         2910         CMP #"/"
1DED- F0 08         2920         BEQ .2          ...Apple-Slash
1DEF- C9 AF         2930         CMP #"/"        <<<I don't know why they do it 3 times>>>
1DF1- F0 04         2940         BEQ .2          <<<Maybe just for the fun of it.......>>>
1DF3- C9 AF         2950         CMP #"/"
1DF5- D0 05         2960         BNE .3          ...not Apple-/
1DF7- A9 BF         2970 .2      LDA #"?"        ...Substitute Apple-?
1DF9- 4C 7D 1E      2980         JMP KEYIN.EXIT
                    2990 *---Map Lower-Case to Upper------
1DFC- C9 E1         3000 .3      CMP #"a"
1DFE- 90 06         3010         BCC .4          ...not lower-case letter
1E00- C9 FB         3020         CMP #"z"+1
1E02- B0 02         3030         BCS .4          ...not lower-case letter
1E04- 29 DF         3040         AND #$DF
                    3050 *---If Apple-Y, make Ctrl-Y------
1E06- C9 D9         3060 .4      CMP #"Y"        Check for Apple-Y
1E08- D0 02         3070         BNE .5          ...no
1E0A- 29 1F         3080         AND #$1F        Changes $D9 to $19, control-Y
                    3090 *---Check for Apple-H------------
1E0C- C9 C8         3100 .5      CMP #"H"        Check for Apple-H
1E0E- D0 4A         3110         BNE .9          ...not Apple-H
1E10- AD 0E 1D      3120         LDA I.1D0E      ???
1E13- D0 05         3130         BNE .6          ...ignore the Apple-H
1E15- AD 3B 0F      3140         LDA X.0F3B
1E18- D0 03         3150         BNE .7          ...go ahead and print
1E1A- 4C 46 1D      3160 .6      JMP KEYIN.ANOTHER.CHAR
                    3170 *---Print the screen-------------
1E1D- 8D 3D 0F      3180 .7      STA X.0F3D
1E20- A9 00         3190         LDA #$00        "OPEN" Printer
1E22- 20 21 1C      3200         JSR PRINTER.DRIVER
1E25- A9 00         3210         LDA #0          For LINE = 0 to 23
1E27- 8D 32 1D      3220         STA KEYIN.APPLE.H.LINE.NUMBER
1E2A- AD 32 1D      3230 .8      LDA KEYIN.APPLE.H.LINE.NUMBER
1E2D- 20 7A 18      3240         JSR COPY.SCRN.LINE.TO.0900
1E30- A9 4F         3250         LDA #79         Print 79 characters from 0900
1E32- 20 21 1C      3260         JSR PRINTER.DRIVER
1E35- 00 09         3270         .DA X.0900
1E37- A9 FE         3280         LDA #$FE        Print CRLF
1E39- 20 21 1C      3290         JSR PRINTER.DRIVER
1E3C- EE 32 1D      3300         INC KEYIN.APPLE.H.LINE.NUMBER  Next Line
1E3F- AD 32 1D      3310         LDA KEYIN.APPLE.H.LINE.NUMBER
1E42- C9 18         3320         CMP #24         Last line yet?
1E44- 90 E4         3330         BCC .8          ...no, keep printing
1E46- A9 FF         3340         LDA #$FF        "CLOSE" Printer
1E48- 20 21 1C      3350         JSR PRINTER.DRIVER
1E4B- 20 E0 1F      3360         JSR CLEAR.KEYBUF
1E4E- AE 33 1D      3370         LDX KEYIN.CURSOR.CH     Put cursor back...
1E51- AC 34 1D      3380         LDY KEYIN.CURSOR.CV     ...but I don't know how
1E54- 20 23 18      3390         JSR MOVE.CURSOR.TO.XY       it could have moved.
1E57- 4C 46 1D      3400         JMP KEYIN.ANOTHER.CHAR      Get another character.
                    3410 *--------------------------------
1E5A- C9 D1         3420 .9      CMP #"Q"        Check for Apple-Q
1E5C- F0 09         3430         BEQ .10
1E5E- C9 D3         3440         CMP #"S"        Check for Apple-S
1E60- D0 0A         3450         BNE .11
1E62- AE A7 0E      3460         LDX X.0EA7
1E65- D0 05         3470         BNE .11
1E67- 85 A4         3480 .10     STA Z.A4        Apple-Q or -S clobbers Z.A4
1E69- 4C 7B 1E      3490         JMP KEYIN.EXIT.ESCAPE
                    3500 *---If Apple-E, change cursor----
1E6C- C9 C5         3510 .11     CMP #"E"        Check for Apple-E
1E6E- D0 0D         3520         BNE KEYIN.EXIT
1E70- AD 76 11      3530         LDA KEYIN.CURSOR.TYPE
1E73- 49 01         3540         EOR #$01        Toggle btwn $00 and $01
1E75- 8D 76 11      3550         STA KEYIN.CURSOR.TYPE
1E78- 4C 46 1D      3560         JMP KEYIN.ANOTHER.CHAR
                    3570 *--------------------------------
                    3580 * (1E7B) 1D39 1E69
                    3590 KEYIN.EXIT.ESCAPE
1E7B- A9 1B         3600         LDA #$1B        Say you typed <ESC>
                    3610 KEYIN.EXIT
1E7D- 85 84         3620         STA Z.84        Store character here too
1E7F- 60            3630         RTS
                    3640 *--------------------------------
                    3650         .EP
```

```
                        3660
                        3670         .PH $1F0A
                        3680 *--------------------------------
1F0A-                   3690 KEYBOARD.TIMEOUT .BS 2
                        3700 *--------------------------------
                        3710 * (1F0C) 1D7E 1D95 1D9F 1DB7 1DC1
                        3720 READ.KEYBOARD
1F0C- 8C 0B 1F          3730         STY KEYBOARD.TIMEOUT+1
1F0F- AC 31 1D          3740         LDY KEYIN.COLUMN.INDEX
1F12- 91 16             3750         STA (AW.BASE),Y
1F14- AD 00 C0          3760 .1      LDA KEYBOARD
1F17- 30 13             3770         BMI .2          ...got a keystroke
1F19- AD 77 11          3780         LDA KEYIN.CURSOR.FLAG
1F1C- F0 F6             3790         BEQ .1          If no cursor, then no time-out either
1F1E- CE 0A 1F          3800         DEC KEYBOARD.TIMEOUT
1F21- D0 F1             3810         BNE .1          ...more time left
1F23- CE 0B 1F          3820         DEC KEYBOARD.TIMEOUT+1
1F26- D0 EC             3830         BNE .1          ...more time left
1F28- A2 00             3840         LDX #$00        timed out, return false
1F2A- F0 11             3850         BEQ .4          ...always
1F2C- 8D 10 C0          3860 .2      STA STROBE      Clear the strobe
1F2F- AE 61 C0          3870         LDX APPLE.OPEN
1F32- 30 07             3880         BMI .3          Apple, leave bit 7 = 1
1F34- AE 62 C0          3890         LDX APPLE.SOLID
1F37- 30 02             3900         BMI .3          Apple, leave bit 7 = 1
1F39- 29 7F             3910         AND #$7F        No Apple, make bit 7 = 0
1F3B- A2 01             3920 .3      LDX #$01        Return TRUE
1F3D- 60                3930 .4      RTS
                        3940 *--------------------------------
                        3950         .EP
                        3960
                        3970         .PH $1FE0
                        3980 *--------------------------------
                        3990 * (1FE0) 1084 181D 1939 1BF1 1DE7 1E4B
                        4000 *   Clear type-ahead buffer
                        4010 *--------------------------------
                        4020 CLEAR.KEYBUF
1FE0- A9 00             4030         LDA #$00
1FE2- 8D 84 11          4040         STA KEYBUF.IN
1FE5- 8D 85 11          4050         STA KEYBUF.OUT
1FE8- 60                4060         RTS
                        4070 *--------------------------------
                        4080         .EP
                        4090
                        4100         .PH $13B2
                        4110 *--------------------------------
                        4120 * (13B2) 137A 1D46
                        4130 *   Check whether any characters are queued up in the
                        4140 *      keyboard buffer.  If so, return TRUE (status .NE.).
                        4150 *      If not, return FALSE (status .EQ.).
                        4160 *--------------------------------
                        4170 CHECK.KEYBUF
13B2- AD 84 11          4180         LDA KEYBUF.IN      If pointers are same, the buffer
13B5- CD 85 11          4190         CMP KEYBUF.OUT          is empty.
13B8- F0 04             4200         BEQ .1             ...it is empty
13BA- A9 01             4210         LDA #$01           ...not empty, return TRUE (.NE.)
13BC- D0 02             4220         BNE .2
13BE- A9 00             4230 .1      LDA #$00
13C0- 60                4240 .2      RTS
                        4250 *---Alternate code to do same----
                        4260 ***     LDA KEYBUF.IN      If pointers are same, the buffer
                        4270 ***     CMP KEYBUF.OUT          is empty.
                        4280 ***     RTS                .NE. if not empty, .EQ. if empty
                        4290 *--------------------------------
                        4300         .EP
                        4310
                        4320         .PH $1823
                        4330 *--------------------------------
                        4340 * (1823) 1024 13A2 1A58 1A89 1B1B 1E54 1E86 1E90 20B6 2B66 2B82
                        4350 *   Move cursor to column (X), line (Y)
                        4360 *      Works by building a string for STRING.DISPLAY
                        4370 *--------------------------------
                        4380 MOVE.CURSOR.TO.XY
1823- 8E 01 0A          4390         STX X.0A01    Build string "05.XX.YY"
1826- 8C 02 0A          4400         STY X.0A02
1829- A9 05             4410         LDA #$05      "GoToXY" code
182B- 8D 00 0A          4420         STA X.0A00
182E- 20 A9 1E          4430         JSR POINT.PSTR.AT.0A00
1831- A9 03             4440         LDA #3        String has 3 characters
1833- 20 D1 14          4450         JSR DISPLAY.STRING
1836- 60                4460         RTS
```

```
                        4470 *--------------------------------
                        4480         .EP
                        4490
                        4500         .PH $1EA9
                        4510 *--------------------------------
                        4520 * (1EA9) 182E 1F85 1FEC 208B 20CC
                        4530 POINT.PSTR.AT.0A00
1EA9- AD 99 10          4540         LDA HANDLE.0A00
1EAC- 85 80             4550         STA PSTR
1EAE- AD 9A 10          4560         LDA HANDLE.0A00+1
1EB1- 85 81             4570         STA PSTR+1
1EB3- 60                4580         RTS
                        4590 *--------------------------------
                        4600         .EP
                        4610
                        4620         .PH $187A
                        4630 *--------------------------------
                        4640 * (187A) 1036 1A81 1E2D
                        4650 *   Used by Apple-H Screen Print function
                        4660 *      (A) = line to be copied
                        4670 *      Copies 80 characters to buffer at $0900
                        4680 *--------------------------------
                        4690 COPY.SCRN.LINE.TO.0900
187A- 20 17 17          4700         JSR BASE.CALC.A
187D- A2 00             4710         LDX #0
187F- A0 00             4720         LDY #0
1881- B1 16             4730 .1      LDA (AW.BASE),Y
1883- 30 06             4740         BMI .2          80-FF
1885- 20 94 1E          4750         JSR MAP.SCRN.CHARS.TO.INTERNAL
1888- 4C 8D 18          4760         JMP .3
188B- 29 7F             4770 .2      AND #$7F
188D- 9D 01 09          4780 .3      STA X.0901,X
1890- 8D 55 C0          4790         STA SCRN.AUX
1893- B1 16             4800         LDA (AW.BASE),Y
1895- 30 06             4810         BMI .4
1897- 20 94 1E          4820         JSR MAP.SCRN.CHARS.TO.INTERNAL
189A- 4C 9F 18          4830         JMP .5
189D- 29 7F             4840 .4      AND #$7F
189F- 9D 00 09          4850 .5      STA X.0900,X
18A2- 8D 54 C0          4860         STA SCRN.MAIN
18A5- E8                4870         INX
18A6- E8                4880         INX
18A7- C8                4890         INY
18A8- C0 28             4900         CPY #40
18AA- 90 D5             4910         BCC .1          ...more on this line
18AC- 60                4920         RTS
                        4930 *--------------------------------
                        4940         .EP
                        4950
                        4960         .PH $1E94
                        4970 *--------------------------------
                        4980 * (1E94) 1885 1897
                        4990 *
                        5000 *   Only called from subroutine which copies
                        5010 *      a screen line to $0900, and only for
                        5020 *      character values $00-7F.
                        5030 *
                        5040 *      00-1F  to  40-5F
                        5050 *      20-3F  no change
                        5060 *      40-5F  to  80-9F  (Mouse Graphics)
                        5070 *      60-7F  no change
                        5080 *--------------------------------
                        5090 MAP.SCRN.CHARS.TO.INTERNAL
1E94- C9 20             5100         CMP #$20
1E96- B0 04             5110         BCS .1          not 00-1F
1E98- 09 40             5120         ORA #$40        Change 00-1F to 40-5F
1E9A- D0 0C             5130         BNE .2          ...always
1E9C- C9 40             5140 .1      CMP #$40
1E9E- 90 08             5150         BCC .2          ...20-3F
1EA0- C9 60             5160         CMP #$60
1EA2- B0 04             5170         BCS .2          ...60-7F
1EA4- 29 BF             5180         AND #$BF        Change 40-5F to 00-1F
1EA6- 09 80             5190         ORA #$80            and then to 80-9F
1EA8- 60                5200 .2      RTS
                        5210 *--------------------------------
                        5220         .EP
                        5230
```

```
                        5240 *-------------------------------
                        5250 *   A Shorter Version of COPY.SCRN.LINE.TO.0900
                        5260 *-------------------------------
                        5270 SC.COPY.SCRN.LINE.TO.0900
0A15- 20 17 17          5280         JSR BASE.CALC.A
0A18- A2 00             5290         LDX #0
0A1A- A0 00             5300         LDY #0
0A1C- 8D 55 C0          5310 .1      STA SCRN.AUX      Even char first
0A1F- 20 2E 0A          5320         JSR GET.MAP.PUT.SCRN.CHAR
0A22- 8D 54 C0          5330         STA SCRN.MAIN     Then odd char
0A25- 20 2E 0A          5340         JSR GET.MAP.PUT.SCRN.CHAR
0A28- C8                5350         INY
0A29- C0 28             5360         CPY #40
0A2B- 90 EF             5370         BCC .1         ...more on this line
0A2D- 60                5380         RTS
                        5390 *-------------------------------
                        5400 *       00-1F  to  40-5F
                        5410 *       20-3F  no change
                        5420 *       40-5F  to  80-9F  (Mouse Graphics)
                        5430 *       60-7F  no change
                        5440 *       80-FF  to  00-7F
                        5450 *-------------------------------
                        5460 GET.MAP.PUT.SCRN.CHAR
0A2E- B1 16             5470         LDA (AW.BASE),Y
0A30- 30 0E             5480         BMI .1         Change 80-FF to 00-7F
0A32- C9 60             5490         CMP #$60       now have 00-7F
0A34- B0 0C             5500         BCS .2         ...60-7F no change
0A36- 69 40             5510         ADC #$40       Change 00-5F to 40-9F
0A38- 30 08             5520         BMI .2         ...40-5F became 80-9F
0A3A- C9 60             5530         CMP #$60
0A3C- 90 04             5540         BCC .2         ...00-1F became 40-5F
0A3E- 29 3F             5550         AND #$3F       Change 60-7F back to 20-3F
0A40- 29 7F             5560 .1      AND #$7F       Change 80-FF to 00-7F
0A42- E8                5570 .2      INX
0A43- 9D 00 09          5580         STA X.0900,X
0A46- 60                5590         RTS
                        5600 *-------------------------------
```

EnterSoft:

Basic-like macros which make the complex simple. Don't re-write that multiplication routine for the hundredth time! Get EnterSoft instead! Do 8/16/32/64 bit Math/Input-Output/Graphics simply without all of the hassles. These routines are a must for the serious programmer who doesn't want to spend all of his/her time trying to re-invent the wheel. DOS 3.3 Version=$30.00, ProDos Version=$30.00, BOTH for only $50.00. GET YOURS TODAY.

A Shape Table Program:

For once! A shape table program which is logically organizied into its componet parts. Each section resides in its own program. The editor, disk access, Hi-Res section; each section is separate. Written almost entirely in Basic, it is easily modified. Not copyprotected! Put them on a Hard Disk, Ram Drive, anywhere! DOS 3.3 Version=$20.00, ProDos Version=$20.00, BOTH for $30.00!

Send Check or Money Order To:

Mark Manning
c/o Simulacron I/Baggy Game
P.O. Box 58598
Webster, TX 77598

ProDos Upgrade for DOS 3.3
EnterSoft Owners = $20.00

Thanks for the letters -
Keep Writing!

Percentage Printer.....Bob Sander-Cederlof

The following program will print the ratio M/N as a percentage, assuming M and N are 24-bit integers. The percentage will be rounded to the nearest integer value, and print as one, two, or three digits followed by the "%" symbol. If M is more than 999% of N, the value printed will be garbled.

The straightforward way to compute this percentage would be to compute P = 100M/N, convert the quotient to decimal, and print it. To accomplish rounding, you could check the remainder after the division: if twice the remainder is greater than or equal to N, increment the quotient.

After a little head-scratching, I discovered another method. I accomplish the division and the conversion to decimal at the same time, and never multiply by 100. For rounding I cheated a little, and added N/256 to M before dividing.

N/256 is a breeze to compute, because it mearely means offseting the addition loop by one byte. However, the CORRECT rounding term would be N/200. The difference between N/256 and N/200 is 56N/51200, or about .0011N. This is "negligible", at least to me. It in effect means that I am rounding up percentages with fractional parts above .4989, instead of just those with fractions .5 or higher. Who will ever notice? Since my main use for this routine is to print what fraction of my hard disk is in use, it will be plenty close enough. (In fact, maybe I don't even need to round at all!)

The division/conversion loop works by using partial division. Each time I call the division loop, I divide M by N; however, I only loop enough times to get the next decimal digit of the quotient. Then I multiply the remainder by ten, so that the next division will generate the next digit. Trust me, it really works!

I added the digit printout to the tail end of the same subroutine which generates the next digit, and put in some logic to suppress leading zeroes. It looks funny if 10% prints out as 010%, so I ignore that leading zero. On the other hand, the logic makes sure 0% does not print as just "%".

To use the subroutine, you first have to store the 24-bit value for N. I wrote a subroutine for this, but you don't necessarily have to use it. If you do use it, load the most significant byte in the A-register, the middle byte in X, and the low byte in Y. I call this loading a 24-bit value into AXY. Then do a JSR STORE.N. In contrast to the usual way you see multi-byte values stored in most 6502 code, I store M and N in High-to-Low order. This made the various loops inside the GET.DIGIT subroutine shorter. Lines 1100-1160 are the STORE.N subroutine. Nothing fancy here!

After storing the N-value, load up the M-value in AXY and do a JSR PERCENT.CALC. The percentage that M is of N will be printed, and the subroutine will return. I wrote a demonstration program, shown in lines 2000-2260. This program

sets N=255 and then prints out all percentages for M = 0 to 255. You can vary the value of N and see different effects.

When you call PERCENT.CALC, lines 1180-1200 store your M-value. Lines 1210-1230 initialize the leading zero flag so that those zeroes will be suppressed. Lines 1240-1350 accomplish the pseudo-rounding I described above. I append another byte to the M-value, which is in effect after the radix point. [Radix point? What's a radix point? In decimal numbers, we call it a decimal point. In binary numbers, we could call it a binary point. In general, it is the demarcation between the integral and fractional parts of a number.] Finally, lines 1370-1410 generate and print three digits of the answer followed by the %-sign.

GET.DIGIT, lines 1430-1800, does all the real work. Lines 1440-1530 subtract N from M until M goes negative. The Y-register counts how many times this takes, less one. Unless M was greater than or equal to 10N, the number in Y will be a value from 0-9, and will be the first or next digit of the percentage. But before printing that digit, I need to modify the remainder.

Lines 1540-1610 add N back one time, so that the remainder is positive. We subtracted once too often, so this fixes things. Then lines 1620-1710 multiply the remainder by 10. Next time GET.DIGIT is called, we will generate the next digit of the percentage.

Lines 1720-1790 print the digit, unless it is a leading zero. If the digit is not zero, it is obviously not a leading zero, so it is printed. Any time a digit gets printed, I store a negative value in my leading zero flag, indicating that any future zeroes cannot be called "leading" (see line 1780). The leading zero flag started out at 2, and each time I test it gets decremented in line 1750. If the first two digits are both zero, it will go negative forcing the third digit to print even it is also zero.

If you are interested, it is very simple to modify this program so that is prints out a rounded percentage in the format xxx.x%, to the nearest tenth of a percent. All we have to do is change the rounding term from N/256 to N/2048, which affects the code in lines 1240-1350, and then add the follow lines:

```
1392    LDA #"."        Print a decimal point
1394    JSR MON.COUT
1396    JSR GET.DIGIT   Print the tenths digit
```

Other simple modifications could change the variable size from 24-bits to 16-bits, 32-bits, or whatever you need.

```
                    1000 *SAVE S.PERCENT.CALC
                    1010 *-------------------------------
                    1020 *   Subroutine for printing M/N as a percentage
                    1030 *   where M and N are 24-bit integers.
                    1040 *      1.  With (AXY)=N, do JSR STORE.N
                    1050 *      2.  With (AXY)=M, do JSR PERCENT.CALC
                    1060 *-------------------------------
FD8E-               1070 MON.CROUT  .EQ $FD8E
FDDA-               1080 MON.PRBYTE .EQ $FDDA
FDED-               1090 MON.COUT   .EQ $FDED
                    1100 *-------------------------------
                    1110 STORE.N
0800- 8D 9B 08      1120         STA N          MOST SIGNIFICANT
0803- 8E 9C 08      1130         STX N+1
0806- 8C 9D 08      1140         STY N+2        LEAST   "
0809- 60            1150         RTS
                    1160 *-------------------------------
                    1170 PERCENT.CALC
080A- 8D 97 08      1180         STA M          MOST SIGNIFICANT
080D- 8E 98 08      1190         STX M+1
0810- 8C 99 08      1200         STY M+2        LEAST   "
                    1210 *---Init Leading Zero Flag-------
0813- A9 02         1220         LDA #2         Start with LZ-flag = 2
0815- 8D A2 08      1230         STA Z
                    1240 *---Add N/256 to Round Result----
0818- AD 9D 08      1250         LDA N+2        Accuracy would demand adding
081B- 8D 9A 08      1260         STA M+3        N/200, but N/256 is close enough.
081E- 18            1270         CLC                  N      N      56N
081F- A2 01         1280         LDX #1               --- = --- + -------
0821- BD 98 08      1290 .1      LDA M+1,X            200    256    200*256
0824- 7D 9B 08      1300         ADC N,X
0827- 9D 98 08      1310         STA M+1,X      And 56/51200 = .0010937 (very small)
082A- CA            1320         DEX
082B- 10 F4         1330         BPL .1
082D- 90 03         1340         BCC .2
082F- EE 97 08      1350         INC M
                    1360 *---Compute & Print Digits-------
0832- 20 40 08      1370 .2      JSR GET.DIGIT      Hundreds digit
0835- 20 40 08      1380         JSR GET.DIGIT      Tens digit
0838- 20 40 08      1390         JSR GET.DIGIT      Units digit
083B- A9 A5         1400         LDA #"%"
083D- 4C ED FD      1410         JMP MON.COUT
                    1420 *-------------------------------
                    1430 GET.DIGIT
0840- A0 FF         1440         LDY #-1        Y will be the quotient
0842- 38            1450         SEC
0843- C8            1460 .1      INY            Increment Quotient
0844- A2 02         1470         LDX #2
0846- BD 97 08      1480 .2      LDA M,X        Subtract Denominator
0849- FD 9B 08      1490         SBC N,X
084C- 9D 97 08      1500         STA M,X
084F- CA            1510         DEX
0850- 10 F4         1520         BPL .2
0852- B0 EF         1530         BCS .1         This goes around once to often...
                    1540 *---Add N back in once-----------
0854- A2 02         1550         LDX #2
0856- BD 97 08      1560 .3      LDA M,X        So we need to add it back once.
0859- 7D 9B 08      1570         ADC N,X
085C- 9D 97 08      1580         STA M,X
085F- 9D 9E 08      1590         STA T,X        Save copy of M in T, to make it
0862- CA            1600         DEX                 easier to multiply by 10.
0863- 10 F1         1610         BPL .3
                    1620 *---Multiply M by 10-------------
0865- 20 8D 08      1630         JSR M.TIMES.2  M = 2 (4M + T )
0868- 20 8D 08      1640         JSR M.TIMES.2  ...now we have 4M
086B- A2 03         1650         LDX #3         ...add T (copy of original M)
086D- BD 97 08      1660 .4      LDA M,X
0870- 7D 9E 08      1670         ADC T,X
0873- 9D 97 08      1680         STA M,X
0876- CA            1690         DEX
0877- 10 F4         1700         BPL .4
0879- 20 8D 08      1710         JSR M.TIMES.2  5M times 2 is 10M
                    1720 *---Print digit if not leading zero---
087C- 98            1730         TYA            digit is quotient from above
087D- D0 05         1740         BNE .5         ...digit not zero, so print it
087F- CE A2 08      1750         DEC Z          Is it a leading zero?
0882- 10 08         1760         BPL .6         ...yes, don't print it.
0884- 09 B0         1770 .5      ORA #"0"       Make it ASCII
0886- 8D A2 08      1780         STA Z          Kill LZ-flag by setting bit 7 = 1
0889- 20 ED FD      1790         JSR MON.COUT Print the digit
088C- 60            1800 .6      RTS
```

```
                      1810 *--------------------------------
                      1820 M.TIMES.2
088D- A2 03           1830         LDX #3        Double 4-byte value in M
088F- 18              1840         CLC
0890- 3E 97 08        1850 .1      ROL M,X
0893- CA              1860         DEX
0894- 10 FA           1870         BPL .1
0896- 60              1880         RTS
                      1890 *--------------------------------
0897-                 1900 M       .BS 4
089B-                 1910 N       .BS 3
089E-                 1920 T       .BS 4
08A2-                 1930 Z       .BS 1         LEADING ZERO FLAG
                      1940 *--------------------------------
                      1950 *    Test Routine for PERCENT.CALC
                      1960 *      For M = 0 to 255
                      1970 *      Print M,PERCENT(M/255)
                      1980 *      Next M
                      1990 *--------------------------------
                      2000 TT
08A3- A9 00           2010         LDA #0
08A5- 8D D6 08        2020         STA TM        Start TM=0
08A8- AA              2030         TAX
08A9- A0 FF           2040         LDY #255      Set N = 255
08AB- 20 00 08        2050         JSR STORE.N
08AE- AC D6 08        2060 .1      LDY TM        Set M to current TM
08B1- 98              2070         TYA
08B2- 29 07           2080         AND #7        If TM Mod 8 = 0, start new line
08B4- D0 03           2090         BNE .2        ...same line
08B6- 20 8E FD        2100         JSR MON.CROUT
08B9- 98              2110 .2      TYA           Get M again
08BA- 20 DA FD        2120         JSR MON.PRBYTE   Print M-value too
08BD- A9 A0           2130         LDA #" "         followed by one blank
08BF- 20 ED FD        2140         JSR MON.COUT
08C2- A9 00           2150         LDA #0
08C4- AA              2160         TAX           Leading bytes of M = 00 00
08C5- 20 0A 08        2170         JSR PERCENT.CALC  print M/N percent
08C8- A9 A0           2180         LDA #" "
08CA- 20 ED FD        2190         JSR MON.COUT      followed by two blanks
08CD- 20 ED FD        2200         JSR MON.COUT
08D0- EE D6 08        2210         INC TM        Next TM
08D3- D0 D9           2220         BNE .1        ...until wraps around to 00
08D5- 60              2230         RTS           Finished
                      2240 *--------------------------------
08D6-                 2250 TM      .BS 1
                      2260 *--------------------------------
```

Another Quick Two-Digit Decimal Printer...Bob Sander-Cederlof

I have written a number of articles in the past about converting values from binary to decimal and printing or displaying them. Usually such routines need to handle large numbers, but sometimes they are limited to single-byte values. And once in a while, you know in advance the value will be between 0 and 99 decimal.

For example, when you are printing the date you know in advance that the day, month, and year numbers are only two digits each. The following program is actually used in one such date printer, to print the day number and year number. For simplicity, it always prints two digits, even if the first digit is a zero. This is the way I want it to be when printing the year, but the day would probably look better without a leading zero.

Lines 1000-1140 in the listing which follows are a test routine which call on my PD subroutine to print every possible value from 00 to 99. Lines 1150 to the end are the PD subroutine. Lines 1240-1290 are not actually assembled, because the variable "blank.fill" is zero. If you change line 1010 to make "blank.fill" equal to 1, lines 1240-1290 will be assembled. Then values less than 10 will print with a leading blank rather than a leading zero.

```
                      1000 *SAVE Q2D.DECIMAL

00-                   1010 blank.fill .eq 0
                      1020 *--------------------------------
FDED-                 1030 COUT   .EQ $FDED
                      1040 *--------------------------------
0800- A0 00           1050 T      LDY #0        For Y = 0 to 99
0802- 98              1060 .1     TYA           A = Y
0803- 20 14 08        1070        JSR PD        Print two digits decimal
0806- A9 A0           1080        LDA #" "      Print two spaces
0808- 20 ED FD        1090        JSR COUT
080B- 20 ED FD        1100        JSR COUT
080E- C8              1110        INY           Next Y
080F- C0 64           1120        CPY #100
0811- 90 EF           1130        BCC .1
0813- 60              1140        RTS           Finished!
                      1150 *--------------------------------
0814- A2 AF           1160 PD     LDX #"0"-1    Start with ASCII zero-1
0816- 38              1170        SEC           Set up subtraction
0817- E8              1180 .1     INX           Increment ten's digit
0818- E9 0A           1190        SBC #10       Take out ten
081A- B0 FB           1200        BCS .1        Still more tens
081C- 69 BA           1210        ADC #"0"+10   Add back one ten, and make ASCII
081E- 48              1220        PHA           Save unit's digit
081F- 8A              1230        TXA           Get ten's digit
                      1240    .do blank.fill
                      1250    cmp #"0"          If these are assembled, print
                      1260    bne .2            00-09 as " 0" through " 9"
                      1270    lda #" "
                      1280 .2
                      1290    .fin
0820- 20 ED FD        1300        JSR COUT      Print ten's digit
0823- 68              1310        PLA           Get unit's digit
0824- 4C ED FD        1320        JMP COUT      and print it
                      1330 *--------------------------------
```

New Versions of S-C Word Processor.....Bob Sander-Cederlof

A long time ago I wished there were some sort of word processor for the Apple II. Paul Lutus wrote AppleWriter, and I bought a copy. It was limited to a 40-column display, and only showed upper-case on the screen, due to limitations in the old Apple II and II Plus machines. It cost $50, and that seemed like a pretty good price.

After a while I found out about the Paymar Lower-Case Adapter, and added lower-case display to the screen. Some patches inside AppleWriter made it work with the adapter. Then the shift-key mod was invented, and I installed that also. I started hoping for even more....

So, I wrote my own word processor, based on AppleWriter's features, and using some of the internal techniques Paul Lutus developed which made AppleWriter faster than any other word processors available. I simplified the editing commands, speeded up and enhanced a lot of the features, and significantly shortened the code. I gave it the ability to use standard text files, with blazingly fast disk load and save. It gradually grew into a product, which we sold with all the source code for $50. More than a pretty good price.

However, the S-C Word Processor was still limited to a 40-column display. Bob Deen, a high school student at the time, was doing some programming work for us. He did a lot of work on our Cross Assemblers, for example. He was also using the S-C Word Processor a lot, so I asked him to make an 80-column version for the Apple //e. He succeeded, and also added "widow" and "orphan" protection. (See the article in AAL, July, 1984.) By the way, Bob Deen is now a computer scientist at Jet Propulsion Laboratories in Pasadena, California, doing image enhancement software.

Meanwhile, Apple made a widow out of DOS 3.3 by bringing out the ProDOS system. The S-C Word Processor was still tied to DOS, partly because of my fancy disk I/O and the catalog-menu system. Then last December Bob Gardner sent us a ProDOS version! (Bob, who lives in Washington state, has been a loyal customer and friend since at least 1983.) He did a terrific job, making the ProDOS version even better than the DOS one. The only drawback was that it only worked in 40-columns. Well, in March he sent an 80-column version.

Now, still for only $50, you get both DOS and ProDOS versions which work in both 40- and 80-columns. To use the 80-column versions you have to have an Apple //e, //c, or IIgs. The 40-column versions will work on those or older Apples, but the older Apples do need lower-case display and shift-key mods.

I wouldn't want you to think the S-C Word Processor has all the features of the Word Perfect, or other such major products. No, it is not that sophisticated. But it does have all the basic features we need for everyday work, it is very fast, and you get all the source code so you can personalize it.

Some of you have done some extensive personalizing already. Horst Schneider, a retired businessman in Denver, modified it to become a part of his business management package, adding mail merge and other features along the way. Larry Skutchan, who works at American Printing House for the Blind, made a talking version which works with Street Electronics' "Echo" speech synthesizers. If you are interested in either of these, I could put you in touch with Horst or Larry.

A curious bit of history: all three of the programmers who have made the major contributions to SCWP are named Bob! I guess we could give it the nickname of the Three-Bob Word Processor! No, it is inexpensive, but we do charge more than three shillings.

If you already have an earlier version of the S-C Word Processor and would like to get an update to the latest, send $7.50 (or only $5 if you only want the ProDOS disk).

Printing the ProDOS Date and Time.....Bob Sander-Cederlof

ProDOS-8 stores the date and time information from in four bytes in the Global Page starting at $BF90:

```
$BF90:  MMMDDDDD   Low-order bits of Month, Day (1-31)
$BF91:  YYYYYYYM   Year (0-99), high bit of Month
$BF92:  00mmmmmm   Minute (0-59)
$BF93:  00hhhhhh   Hour (0-23)
```

The following subroutine, lines 1000-1590, will print out the date in the form DD-MMM-YY. Lines 1600-1800 are an alternative method for printing out the 3-letter month name abbreviation. Lines 1810-1910 print the time in the form hh:mm.

```
                    1020 *--------------------------------
                    1030 *   Subroutine to print date from ProDOS Global Page
                    1040 *      in form DD-MMM-YY.
                    1050 *   Two different methods for printing the 3-letter month
                    1060 *      name are shown, with month-name table in normal and
                    1070 *      transposed order.
                    1080 *--------------------------------
BF90-               1090 DATE   .EQ $BF90,BF91   Date in form: MMMDDDDD, YYYYYYYM
                    1100 *                       Time in form: 00mmmmmm, 00Ohhhhh
FDED-               1110 COUT   .EQ $FDED
                    1120 *--------------------------------
                    1130 PRINT.DATE
0800- AD 90 BF      1140        LDA DATE     Get MMMDDDDD
0803- 29 1F         1150        AND #$1F     Isolate Day of Month
0805- 20 33 08      1160        JSR PD       Print the day number
0808- A9 AD         1170        LDA #"-      Print a dash
080A- 20 ED FD      1180        JSR COUT
                    1190 *----PRINT MONTH FROM TABLE------
080D- AD 91 BF      1200        LDA DATE+1   Get YYYYYYYM
0810- 4A            1210        LSR          High bit of Month-number into Carry
0811- 48            1220        PHA          Save 0YYYYYYY on stack
0812- AD 90 BF      1230        LDA DATE     Get MMMDDDDD
0815- 6A            1240        ROR          MMMMDDDD
0816- 4A            1250        LSR          0MMMMDDD
0817- 4A            1260        LSR          00MMMMDD
0818- 4A            1270        LSR          000MMMMD
0819- 4A            1280        LSR          0000MMMM   Month number (1-12)
081A- AA            1290        TAX
081B- BD 45 08      1300        LDA MONTH.TBL.1-1,X  1st letter
081E- 20 ED FD      1310        JSR COUT
0821- BD 51 08      1320        LDA MONTH.TBL.2-1,X  2nd letter
0824- 20 ED FD      1330        JSR COUT
0827- BD 5D 08      1340        LDA MONTH.TBL.3-1,X  3rd letter
082A- 20 ED FD      1350        JSR COUT
082D- A9 AD         1360        LDA #"-      Print dash
082F- 20 ED FD      1370        JSR COUT
                    1380 *----PRINT YEAR-----------------
0832- 68            1390        PLA          GET 0YYYYYYY FROM STACK
                    1400 *---Fall into PD subroutine------
0833- A2 AF         1410 PD     LDX #"0"-1   Start with ASCII zero-1
0835- 38            1420        SEC          Set up subtraction
0836- E8            1430 .1     INX          Increment ten's digit
0837- E9 0A         1440        SBC #10      Take out ten
0839- B0 FB         1450        BCS .1       Still more tens
083B- 69 BA         1460        ADC #"0"+10  Add back one ten, and make ASCII
083D- 48            1470        PHA          Save unit's digit
083E- 8A            1480        TXA          Get ten's digit
083F- 20 ED FD      1490        JSR COUT     Print ten's digit
0842- 68            1500        PLA          Get unit's digit
0843- 4C ED FD      1510        JMP COUT     and print it
                    1520 *--------------------------------
                    1530        .MA AS
                    1540        .AS -/]1/
                    1550        .EM
                    1560 *--------------------------------
0846-               1570 MONTH.TBL.1 >AS "JFMAMJJASOND"
0852-               1580 MONTH.TBL.2 >AS "AEAPAUUUECOE"
085E-               1590 MONTH.TBL.3 >AS "NBRRYNLGPTVC"
```

```
                1600 *--------------------------------
                1610 ALTERNATIVE.MONTH.PRINTER
086A- AD 91 BF  1620         LDA DATE+1    GET YYYYYYYM
086D- 4A        1630         LSR           M INTO CARRY
086E- AD 90 BF  1640         LDA DATE      GET MMMDDDDD
0871- 6A        1650         ROR           MMMMDDDD
0872- 4A        1660         LSR           0MMMMDDD
0873- 4A        1670         LSR           00MMMMDD
0874- 4A        1680         LSR           000MMMMD
0875- 4A        1690         LSR           0000MMMM
0876- 8D 8B 08  1700         STA TEMP      Multiply month number by 3
0879- 0A        1710         ASL
087A- 6D 8B 08  1720         ADC TEMP
087D- AA        1730         TAX           Index is 3,6,9,...
087E- A0 03     1740         LDY #3        Print 3 consecutive letters
0880- BD 89 08  1750 .1      LDA MONTH.TABLE-3,X
0883- 20 ED FD  1760         JSR COUT
0886- E8        1770         INX           Next letter
0887- 88        1780         DEY
0888- D0 F6     1790         BNE .1
088A- 60        1800         RTS           Finished
                1810 *--------------------------------
088B-           1820 TEMP    .BS 1
088C-           1830 MONTH.TABLE >AS "JANFEBMARAPRMAYJUNJULAUGSEPOCTNOVDEC"
                1840 *--------------------------------
                1850 PRINT.TIME
08B0- AD 93 BF  1860         LDA DATE+3    Get 00hhhhhh
08B3- 20 33 08  1870         JSR PD
08B6- A9 BA     1880         LDA #":"
08B8- 20 ED FD  1890         JSR COUT
08BB- AD 92 BF  1900         LDA DATE+2    Get 00mmmmmm
08BE- 4C 33 08  1910         JMP PD
```

The value stored in the Global Page may not be current. It is automatically updated every time you close or flush a file, or you can force it to be updated by using the MLI call shown in lines 1920-end. If you have looked into the Global Page description in the manuals, you may have noticed that $BF06 is a vector to the date/time update code. Don't try to use it directly unless you are sure the Language Card is properly switched before and after the call. The best way is to use the MLI call, as I did.

```
                1920 *--------------------------------
                1930 UPDATE.DATE.AND.TIME
08C1- 20 00 BF  1940         JSR $BF00     MLI ENTRY POINT
08C4- 82 00 00  1950         .DA #$82,0000     GET DATE/TIME, NO PARMS
08C7- 60        1960         RTS
                1970 *--------------------------------
```

Apple Assembly Line (ISSN 0889-4302) is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $24 per year in the USA, Canada, and Mexico, or $36 in other countries. Back issues are $1.80 each for Volumes 1-7, $2.40 each from later Volumes (plus postage). A subscription to the newsletter with a Monthly Disk containing all program source code and article text is $64 per year in the USA, Canada and Mexico, and $90 to other countries.